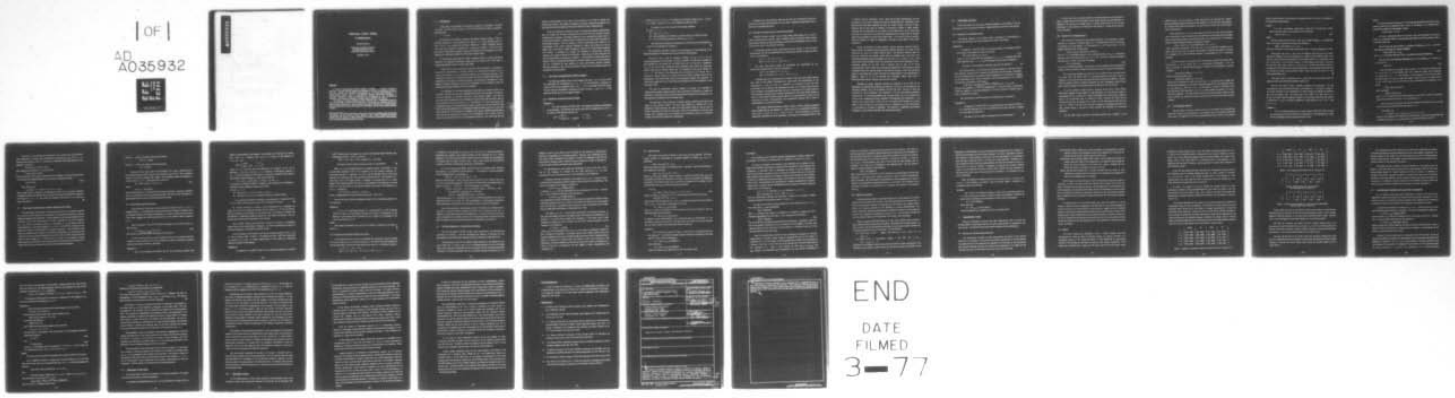
AD-A035 932 CARNEGIE-MELLON UNIV PITTSBURGH PA DEPT OF COMPUTER --ETC F/G 9/2
ASYNCHRONOUS ITERATIVE METHODS FOR MULTIPROCESSORS.(U)
NOV 76 G M BAUDET N00014-76-C-0370

UNCLASSIFIED NL

| OF |
AD
A035932



END

DATE
FILMED
3-77

Asynchronous Iterative Methods

for Multiprocessors

Gérard M. Baudet

Department of Computer Science
Carnegie-Mellon University
Pittsburgh, Pennsylvania 15213

November, 1976

Abstract

A class of asynchronous iterative methods is presented for solving a system of equations. Existing iterative methods are identified in terms of asynchronous iterations, and new schemes are introduced corresponding to a parallel implementation on a multiprocessor system with no synchronization between cooperating processes. A sufficient condition is given to guarantee the convergence of any asynchronous iterations, and results are extended to include iterative methods with memory.

Asynchronous iterative methods are then evaluated from a computational point of view, and bounds are derived for the efficiency. The bounds are compared with actual measurements obtained by running various asynchronous iterations on a multiprocessor, and the experimental results show clearly the advantage of purely asynchronous iterative methods.

This research was partly supported by the National Science Foundation under Grant MCS 75-222-55 and the Office of Naval Research under Contract N00014-76-C-0370, NR 044-422 and partly by a Research Grant from the Institut de Recherche d'Informatique et d'Automatique (IRIA), Rocquencourt, France.

1 - Introduction

In this paper we investigate the fixed point problem for an operator F from $\mathbb{R}^n$ into itself: we want to find a vector x in $\mathbb{R}^n$ which satisfies the system of equations represented by

$$x = F(x). \tag{1.1}$$

In [1], Chazan and Miranker introduced the *chaotic relaxation scheme*, a class of iterative methods for solving equation (1.1) where F is a linear operator given by $F(x) = Ax + b$. They showed that iterations defined by a chaotic relaxation scheme converge to the solution of equation (1.1) if and only if $\rho(|A|) < 1$. (If M is a real $n \times n$ matrix, $\rho(M)$ denotes its spectral radius and $|M|$ denotes the non-negative $n \times n$ matrix obtained by replacing the elements of M by their absolute values.)

In [4], Miellou generalized the chaotic relaxation scheme to include non-linear operators and obtained convergence results similar to those of [1] in the case of *contracting operators* (see, for example, [5, p. 433]).

In both [1] and [4], the motivation of defining chaotic relaxation is to account for the parallel implementation of iterative methods on a multiprocessor system so as to reduce communication and synchronization between the cooperating processes. This reduction is obtained by not forcing the processes to follow a predetermined sequence of computations, but simply by allowing a process, when starting the evaluation of a new iterate, to choose dynamically not only the components to be evaluated but also the values of the previous iterates used in the evaluation.

The definition of the chaotic relaxation scheme does not, however, allow for a completely arbitrary choice of the antecedent values used in the evaluation of an iterate. The main restriction is that there must exist a *fixed* positive integer s such that, in carrying out the evaluation of the i-th iterate, a process cannot make use of any value of the components of the j-th iterate if $j < i-s$. For example, if, for some reason (due to the computation itself or to the mutiprocessor system), a process may take an arbitrarily long time to relax the components it is evaluating, the other processes may have to wait until the evaluation by the first process is completed. This requires repeated checking before each step of the iteration and some form of synchronization. This is exactly what we want to avoid because the use of synchronization primitives is time consuming and also

because synchronization forces some of the processors to be idle or implies the switching of context. This creates an unnecessary overhead, reduces the parallelism, and decreases the maximum speed-up we expect to achieve in using a multiprocessor.

In the next section we introduce the class of *asynchronous iterative methods* which does not impose the restriction mentioned above, and we show that existing iterative methods (and, in particular, the chaotic relaxation) can be represented as special cases of asynchronous iterations. Section 3 gives the definition and reviews some properties of *contracting operators*. Then the theorem of section 4 generalizes the results on the convergence of the chaotic relaxation obtained by Chazan and Miranker [1] and by Miellou [4]. This result is further extended, in section 5, to include iterative methods with memory. In section 6, we consider the complexity of asynchronous iterative methods, and we derive bounds on the efficiency. These bounds are then compared with actual measurements of asynchronous iterations. The experimental results, presented in section 7, show a considerable advantage for iterations making no use of synchronization, and this constitutes the best argument for using asynchronous iterative methods. Possible extensions of the results are discussed in section 9, and concluding remarks are presented in the last section.

2 - The class of asynchronous iterative methods

The following notations will be used throughout the paper. If x is a vector of $\mathbb{R}^n$, its components will be denoted by x_i, $i = 1, \dots, n$. To avoid confusion, a sequence of vectors of $\mathbb{R}^n$ will be denoted by $x(j)$, $j = 0, 1, \dots$. If F is an operator of $\mathbb{R}^n$ into itself, $F(x)$ will also be represented in components by $f_i(x)$ or by $f_i(x_1, \dots, x_n)$, $i = 1, \dots, n$. We denote by $\mathbb{N}$ the set of all non-negative integers.

2.1 - Definition of asynchronous iterative methods

Definition 1:

Let F be an operator from $\mathbb{R}^n$ to $\mathbb{R}^n$. An *asynchronous iteration* corresponding to the operator F and starting with a given vector $x(0)$ is a sequence $x(j)$, $j = 0, 1, \dots$, of vectors of $\mathbb{R}^n$ defined recursively by:

$$x_i(j) = \begin{cases} x_i(j-1) & \text{if } i \notin J_j \\ f_i(x_1(s_1(j)), \dots, x_n(s_n(j))) & \text{if } i \in J_j, \end{cases} \tag{2.1}$$

where $\mathcal{J} = \{ J_j \mid j = 1, 2, \ldots \}$ is a sequence of non-empty subsets of $\{1, \ldots, n\}$ and $\mathcal{S} = \{ (s_1(j), \ldots, s_n(j)) \mid j = 1, 2, \ldots \}$ is a sequence of elements in $\mathbb{N}^n$.

In addition, $\mathcal{J}$ and $\mathcal{S}$ are subject to the following conditions:

for each $i = 1, \ldots, n$

(a) $s_i(j) \leq j-1$, $j = 1, 2, \ldots$,

(b) $s_i(j)$, considered as a function of j, tends to infinity as j tends to infinity,

(c) i occurs infinitely many often in the sets J_j, $j = 1, 2, \ldots$.

An asynchronous iteration corresponding to F, starting with $x(0)$ and defined by $\mathcal{J}$ and $\mathcal{S}$ will be denoted by $(F, x(0), \mathcal{J}, \mathcal{S})$. ∎

An asynchronous iteration $(F, x(0), \mathcal{J}, \mathcal{S})$ may be thought of as corresponding to the following sequence of computations on an asynchronous multiprocessor.

Assume we have a pool of processors available. Let t_j, $j = 1, 2, \ldots$, be an increasing sequence of time instants. At time t_j processor P is idle and is assigned to the evaluation of the iterate $x(j)$, $x(j)$ differs from $x(j-1)$ by the set of components $\{ x_i \mid i \in J_j \}$ and P starts computing these components using values of components known from previous iterates, namely the r-th component of the $s_r(j)$-th iterate, for $r = 1, \ldots, n$. The choice of the components may be guided by any criterion, and, in particular, a natural criterion is to pick up the most recently available values of the components. This scheme does need any synchronization between the processes. At some time t_k, later on $(k > j)$, P will finish its computations and will be assigned to a new evaluation: $x(k)$.

The use of asynchronous iterative methods is obviously not restricted to multiprocessor systems, and the scheme is also well suited for execution on a network of computers, in particular, when the communication between elements of the network is not too expensive as opposed to the computation itself.

We notice that, in the evaluation of an iterate, nothing is imposed on the use of the values of the previous iterates. The only thing required, by condition (b) of the definition, is that, eventually, the values of an early iterate cannot be used any more in further evaluations and more and more recent values of the components have to be used instead. On a multiprocessor, this condition can be satisfied as long as no processor crashes (and eventually completes its computation).

Condition (a) of the definition states the fact that only components of previous iterates can be used in the evaluation of a new iterate. Condition (c) guarantees that no component be abandoned forever.

2.2 - Examples and particular cases of asynchronous iterations

Classical iterative methods: point or block Jacobi, Gauss-Seidel, etc., as well as others introduced more recently: *chaotic relaxation scheme* [1], *periodic chaotic scheme* [2], *itération chaotique à retards* [4], *itération chaotique série-parallèle* [6], can all be seen as particular cases of asynchronous iterations.

For example, the point-Jacobi method defined on the operator F with the initial approximation $x(0)$ can be represented by the asynchronous iteration $(F,x(0),\mathcal{J},\mathcal{S})$ where $\mathcal{J}$ and $\mathcal{S}$ are defined by:

$$J_j = \{ 1, \dots, n \} \text{ for } j = 1, 2, \dots,$$
$$s_i(j) = j-1 \text{ for } j = 1, 2, \dots \text{ and } i = 1, \dots, n .$$

The same point-Jacobi method can equivalently be represented by the asynchronous iteration where $\mathcal{J}$ and $\mathcal{S}$ are defined by:

$$J_j = \{ 1 + (j-1 \bmod n) \} \text{ for } j = 1, 2, \dots,$$
$$s_i(j) = n \lfloor (j-1)/n \rfloor \text{ for } j = 1, 2, \dots \text{ and } i = 1, \dots, n .$$

Although those two representations correspond to the same point-Jacobi method, they differ by the implicit information they contain about the decomposition of the computations. In the first case, all components are evaluated at once and this, presumably, will be done by one computational process. In the second case, however, each component is evaluated separately, and up to n processes can be used to perform the evaluations. Between the two extreme representations of the point-Jacobi method, in terms of asynchronous iterations, several others can be proposed, each of which can be interpreted in terms of decomposition into computational processes and in terms of implementation by concurrent processes.

The iterative method proposed by Robert, Charnay and Musy (*itération chaotique série-parallèle* [6]) can be obtained as a special case of an asynchronous iteration in which $s_i(j) = j-1$ (for all $i = 1, \dots, n$ and $j = 1, 2, \dots$). This corresponds to a strictly sequential computation of sets of components. The choice of the components within a set

is arbitrary and the calculations of their values can be done simultaneously but the evaluation of a new set of components cannot be started before all components of the previous set have been computed and their new values relaxed. The goal of their research was to show that, for example, in the iterative solution of linear systems resulting from the application of the method of finite differences to partial differential equations, it is possible to concentrate the computations more on those points of the grid where the convergence is slower than on other nodes. This is not the case with ordinary iterative methods for which any component is iterated as many times as any other component.

Chazan and Miranker [1] have proposed a *chaotic relaxation scheme* to solve a linear system. Our definition of an asynchronous iterative method is very similar to the definition they give for a chaotic iterative scheme. Our definition, however, does not have the restriction they impose, namely (with our notations) that $j-s_i(j)$ has to be uniformly bounded by some fixed integer, say s, (for all $i = 1, \ldots, n$ and $j = 1, 2, \ldots$). This means that, in the evaluation of the j-th iterate, only values of the components of the s preceding iterates can be used. From a practical point of view, in an actual implementation of such scheme on an asynchronous multiprocessor, this requires a strong assumption about the relative speeds of the different processors, about the scheduling policy of the supervisory system, and about the implementation of the computations in general. There is no way to guarantee this assumption without some form of synchronization (which is precisely what we want to avoid).

Although all chaotic relaxation methods (as presented in [1] or [4]) can be identified as asynchronous iterations, the converse is not true as is illustrated by the following example. Let F be an operator from $\mathbb{R}^2$ into itself. Assume we have two processes P_1 and P_2 attached to the evaluations of the first and second components, respectively. To avoid synchronization, the processes always use in an evaluation the values of the components currently available at the begining of the computation. If we assume that it always takes 1 unit of time for P_1 to perform the evaluation of x_1 and it takes k units of time for P_2 to perform the k-th evaluation of x_2, then the quantity $j - s_2(j)$ grows as $\sqrt{j}$ which is unbounded. This iteration is a legitimate asynchronous iteration, it is not, however, allowed in the setting of [1] and [4].

3 - Contracting operators

In the next section we shall give a sufficient condition on the operator F for the convergence of any asynchronous iteration. Needed definitions are given in this section.

3.1 - Lipchitzian and contracting operators

Contracting operators to be defined below correspond to *P-contractions* in [5, p. 433], and the notion was used to obtained the results of [4] and [6].

Definition 2:

An operator F from $\mathbb{R}^n$ to $\mathbb{R}^n$ is a *Lipchitzian operator* on a subset D of $\mathbb{R}^n$ if there exists a non-negative $n \times n$ matrix A such that:

$$|F(x)-F(y)| \leq A|x-y| \ , \ \forall \ x, y \in D \ , \tag{3.1}$$

where, if z is a vector of $\mathbb{R}^n$ with components z_i, $i = 1, \ldots, n$, $|z|$ denotes the vector with components $|z_i|$, $i = 1, \ldots, n$, and the inequality holds for every component.

The matrix A will be called a *Lipchitzian matrix* for the operator F. ∎

From this definition we can see that any Lipchitzian operator is continuous and, in fact, uniformly continuous on D. However this definition is too broad and, in particular, we are not guaranteed of the existence and the uniqueness of a fixed point as is shown by the following example. Take the operator F from $\mathbb{R}$ to $\mathbb{R}$ defined by $F(x) = \sqrt{x^2+a^2}$, this operator is Lipchitzian on $\mathbb{R}$ because

$$|F(x)-F(y)| = |(x-y)[(x+y)/(\sqrt{x^2+a^2} + \sqrt{y^2+a^2})]| \leq |x-y| \ , \ \forall \ x, y \in \mathbb{R} \ .$$

However, the equation $x = \sqrt{x^2+1}$ (corresponding to $a = 1$) has no solution. On the other hand, the equation $x = |x|$, (corresponding to $a = 0$) has an infinity of solutions, and, in fact, a continuum of solutions.

We will, therefore, restrict ourselves to the following class of operators.

Definition 3:

An operator F from $\mathbb{R}^n$ to $\mathbb{R}^n$ is a *contracting operator* on a subset D of $\mathbb{R}^n$ if it is a Lipchitzian operator on D with a Lipchitzian matrix A such that $\rho(A) < 1$ (where $\rho(A)$ is the spectral radius of A).

The matrix A will be called a *contracting matrix* for the operator F. ∎

The fact that, unlike Lipchitzian operators, contracting operators are guaranteed to have a unique fixed point in the subset D can be easily derived from the definition. In addition, if we assume, for example, that D is closed and that $F(D) \subset D$, we are also guaranteed of the existence of a fixed point in the subset D. A proof can be found in [5, pp. 433-434].

3.2 - Examples of contracting operators

We could have considered a more general definition for asynchronous iterative methods by introducing a relaxation factor $\omega > 0$. This would simply consist of replacing, in equations (2.1), the operator F by the operator $F_\omega = \omega F + (1-\omega)E$, where E is the identity operator of $\mathbb{R}^n$. It follows that

$$|F_\omega(x)-F_\omega(y)| \leq \omega|F(x)-F(y)| + (1-\omega)|x-y| ,$$

and, if F is a contracting operator with a contracting matrix A, F_ω is a Lipchitzian operator with the Lipchitzian matrix $A_\omega = \omega A + |1-\omega|I$. The matrix A being non-negative we have $\rho(A_\omega) = \omega\rho(A) + |1-\omega|$, and, if we choose

$$0 < \omega < 2/[1+\rho(A)] , \tag{3.2}$$

F_ω is also a contracting operator. In particular, as long as condition (3.2) is satisfied, the results of the next section also apply to asynchronous iterative methods with relaxation.

Let F be a linear operator given by $F(x) = Ax + b$, where A is an $n \times n$ matrix and b is a vector of $\mathbb{R}^n$. We observe that F is a contracting operator if and only if $\rho(|A|) < 1$. Therefore, in the case of linear operators, the notion of contracting operators coincides with the property stated by Chazan and Miranker for their convergence result [1], and their result will appear as a particular case of the theorem of the next section.

If we now consider a linear system of equations derived from a linear elliptic differential equation by the method of finite differences, we note that the system is represented by $Ax = b$, where b is a vector of $\mathbb{R}^n$ obtained from the boundary conditions and A is an $n \times n$ M matrix (see, for example, [7, p. 85]). Therefore the system can be written into the form of equation (1.1) in which F is the contracting operator given by $F(x) = (I - D^{-1}A)x + D^{-1}b$, where D is the matrix composed of the diagonal elements of A. This example shows, in the case of linear operators, the importance of contracting operators.

On the other hand, non-linear contracting operators, too, constitute a very

important class. A first example is directly derived from the previous one. Elliptic partial differential equations, obtained by the addition of a small non-linear perturbation to a linear partial differential equation, can also be shown to give rise to (non-linear) contracting operators.

More important, if G is a non-linear operator from $\mathbb{R}^n$ into itself with the simple root ξ, superlinear iterative methods have been devised to find the root ξ of G, provided that an initial approximation $x(0)$ sufficiently close to ξ is already known. For example, Newton iterative method generates the sequence of iterates

$$x(i+1) = F(x(i)) = x(i) - [G'(x(i))]^{-1}G(x(i)), \text{ for } i = 0, 1, \dots,$$

which converges quadratically to the root ξ of G. In this particular example, we can easily derive, under usual assumptions (for example, G' satisfies some Lipchitz condition in a neighbor of ξ), that the Newton operator F corresponding to G is a contracting operator.

In fact this result is very general. Let F be an operator from $\mathbb{R}^n$ into itself with a fixed point ξ. If we assume that F is continuously differentiable in the set $D_r = \{ x \mid \|x-\xi\| < r \}$ and that the derivative F' vanishes at ξ and satisfies a Lipchitz condition

$$\|F'(x)-F'(z)\| \leq M\|x-y\|, \quad \forall x, y \in D_r,$$

then it can be easily shown that

$$\|F(x)-F(y)\| \leq 2Mr\|x-y\|, \quad \forall x, y \in D_r.$$

Therefore, by choosing the vector norm $\|x\| = |x_1| + \dots + |x_n|$ (which only changes the constant M), the operator F is certainly a Lipchitzian operator with the Lipchitzian matrix $A = [a_{ij}]$ where $a_{ij} = 2Mr$, for $i, j = 1, \dots, n$. In particular, if we know a sufficiently close approximation to the fixed point ξ (i. e., if r is small enough), the operator F is also a contracting operator. This shows that the class of contracting operators contains, under weak conditions, all iterative functions occurring in the classical superlinear iterative methods.

4 - Convergence theorem

Before stating a sufficient condition ensuring the convergence of an asynchronous iteration, we give a characterization of a non-negative matrix with spectral radius less than unity. An algebraic proof of this characterization can be found in [1, p. 218], a

shorter proof, based on the continuity of the spectral radius of a matrix as a function of its coefficients, is given below.

Lemma:

Let A be a non-negative square matrix. Then $\rho(A) < 1$ if and only if there exists a positive scalar ω and a positive vector v such that:

$$Av \leq \omega v \quad \text{and} \quad \omega < 1 . \tag{4.1}$$

Proof:

We first assume that (4.1) holds. In this case we note that $\|A\|_v \leq \omega < 1$, where the matrix norm $\|.\|_v$ is induced by the vector norm defined by:

$$\|x\|_v = max\{\ |x_i|/v_i \mid i = 1, \ldots, n\ \} .$$

Therefore the matrix A is convergent which implies $\rho(A) < 1$ (see, for example, [7, p. 13]).

Now assume that $\rho(A) < 1$. Let t be a non-negative scalar and A_t be the matrix obtained by adding t to all null coefficients of A. Clearly, for any positive vector x, we have $Ax \leq A_t x$. On the other hand, $\rho(A_t)$ is a continuous function of t. In particular, since $A_0 = A$ and $\rho(A) < 1$, we can always choose $t > 0$ small enough so that $\rho(A_t) < 1$ (in fact, we also have $\rho(A) \leq \rho(A_t)$). Then let $\omega = \rho(A_t)$. As $A_t > 0$, from Perron's theorem (see, for example, [7, p. 30]), there exists a positive eigenvector v corresponding to the eigenvalue ω. The positive scalar ω and the positive vector v verify $Av \leq A_t v = \omega v$ with $\omega < 1$. And this completes the proof. ∎

This proof shows, in particular, that $\omega \geq \rho(A)$. But, we also see easily that the positive scalar ω can be chosen arbitrarily close to $\rho(A)$.

We are now able to state a sufficient condition on the operator F for the convergence of any asynchronous iteration corresponding to F. This result is similar to the results obtained for the convergence of a chaotic iteration by Chazan and Miranker [1] and by Miellou [4]. The proof given here follows the same idea as in [1, pp. 217-218], it does not depend, however, on the assumption that (with the notation of definition 1) $j - s_i(j)$ is uniformly bounded by a fixed integer, for any $j = 0, 1, \ldots$ and $i = 1, \ldots, n$.

Theorem 1:

If F is a contracting operator on a closed subset D of $\mathbb{R}^n$ and if $F(D) \subset D$, then any asynchronous iteration $(F, x(0), \mathcal{J}, \mathcal{S})$ corresponding to F and starting with a vector $x(0)$ in D converges to the unique fixed point of F in D.

Proof:

Let ξ be the unique fixed point of F. By considering the operator $F(x+\xi)-\xi$, we may assume, without loss of generality, that $\xi = F(\xi) = 0$. By setting $y = \xi$ in equation (3.1), the Lipchitz condition on the operator F gives:

$$|F(x)| \leq A|x| , \quad \forall x \in D .$$

Let A be a contracting matrix for F and let ω and v be as defined in the lemma. Since v is a positive vector, for any starting vector $x(0)$ we can find a positive scalar α such that $|x(0)| \leq \alpha v$.

We will show that we can construct a sequence of indices j_p, $p = 0, 1, \ldots$, such that the sequence of iterates of $(F, x(0), J, \delta)$ satisfies:

$$|x(j)| \leq \alpha \omega^p v \quad \text{for} \quad j \geq j_p . \tag{4.2}$$

As $0 < \omega < 1$, this shows that $x(j) \to 0$ as $j \to \infty$ and this will prove the theorem.

We first show that inequality (4.2) holds for $p = 0$ if we choose $j_0 = 0$. That is, for $j \geq 0$ we have:

$$|x(j)| \leq \alpha v . \tag{4.3}$$

From the choice of α, inequality (4.3) is true for $j = 0$. Assume, for induction, that it is true for $0 \leq j < k$ and consider $x(k)$. Let z denote the vector with components $z_i = x_i(s_i(k))$, for $i = 1, \ldots, n$. From definition 1, the components of $x(k)$ are given either by $x_i(k) = x_i(k-1)$ if $i \notin J_k$, in which case $|x_i(k)| = |x_i(k-1)| \leq \alpha v_i$, or by $x_i(k) = f_i(z)$ if $i \in J_k$. In this latter case, we note that, as $s_i(k) < k$ (condition (a) of definition 1), we have:

$$|F(z)| \leq A|z| \leq \alpha A v \leq \alpha \omega v$$

and in particular:

$$|x_i(k)| = |f_i(z)| \leq \alpha \omega v_i .$$

As $0 < \omega < 1$, in this case too we obtain $|x_i(k)| \leq \alpha v_i$ and (4.3) is proved by induction, which shows that (4.2) is true for $p = 0$ if we choose $j_0 = 0$.

Now assume that j_p has been found and that inequality (4.2) holds for $0 \leq p < q$. We want to find j_q and show that (4.2) also holds for $p = q$.

First define r by

$$r = \min\{ k \mid \forall j \geq k \;\; s_i(j) \geq j_{q-1} , \;\; \text{for} \;\; i = 1, \ldots, n \} .$$

We see, from condition (b) of definition 1, that this number exists, and we note that, from condition (a), we have $r > j_{q-1}$ which shows, in particular, that $|x(r)| \leq \alpha \omega^{q-1} v$.

Then take $j \geq r$ and consider the components of $x(j)$. As above, let z be the vector with components $z_i = x_i(s_i(j))$. From the choice of r, we have $s_i(j) \geq j_{q-1}$, for $i = 1, \ldots, n$, and this shows that $|z| \leq \alpha\omega^{q-1}v$. In particular, in using the contracting property of the operator F we obtain:

$$|F(z)| \leq A|z| \leq \alpha\omega^{q-1}Av \leq \alpha\omega^{q}v .$$

This inequality shows that, if $i \in J_j$, $x_i(j)$ satisfies:

$$|x_i(j)| = |f_i(z)| \leq \alpha\omega^{q}v_i .$$

On the other hand, if $i \notin J_j$ the i-th component is not modified. Therefore, as soon as the i-th component is updated between the r-th and the j-th iteration we have:

$$|x_i(j)| \leq \alpha\omega^{q}v_i . \qquad (4.4)$$

Now, define j_q as:

$$j_q = \min\{\, j \mid j \geq r \text{ and } \{1, \ldots, n\} = J_r \cup \ldots \cup J_j \,\}$$

(this number exists by condition (c) of definition 1), then for any $j \geq j_q$ every component is updated at least once between the r-th and the j-th iteration and therefore inequality (4.4) holds for $i = 1, \ldots, n$. This shows that inequality (4.2) holds for $p = q$ and this proves the theorem. ∎

5 - The class of asynchronous iterative methods with memory

The idea behind the definition of asynchronous iterations, as presented in section 2, is to allow, in the evaluation of $F(x)$, different (and independent) processes to compute different subsets of the components. This corresponds to a natural decomposition for the evaluation of $F(x)$ when the operator F is known explicitely by the set of functions $f_1, \ldots, f_n$. This is not, however, always so. For example, if F is the Newton operator corresponding to a non-linear operator G, i. e.: $F(x) = x - [G'(x)]^{-1}G(x)$, usually only the operator G is given and the operator F is not known explicitely. In this particular case, when two processors are available, a more natural decomposition, as proposed by Kung in [3], is to have one process computing the value of G' while the other process uses this value for the evaluation of F. More precisely, if x and y are two global variables containing the current values of the iterate and of the reciprocal of the derivative of G, respectively, the two processes correspond to the two following programs.

Process 1: while (termination criterion not satisfied)

do $x \leftarrow x - \gamma x G(x)$.

Process 2: while (termination criterion not satisfied)

do $\gamma \leftarrow [G'(x)]^{-1}$.

Starting with the initial values $x(0)$ and $[G'(x(0))]^{-1}$ for x and γ respectively, the two processes execute their programs asynchronously and use for x and γ whatever values are currently available when needed. They implicitely define the sequence of iterates $x(j)$, for $j = 0, 1, \ldots$, through formulas of the form:

$$x(j) = H[x(j-1), x(k_j)], \text{ with } k_j \leq j-1, \tag{5.1}$$

where

$$H(x,\gamma) = x - [G'(\gamma)]^{-1} G(x).$$

This iteration, however, is not allowed in the setting of definition 1, because, in equation (5.1), $x(j)$ is defined in terms of two previous iterates. And this motivates the need for a generalization of the class of asynchronous iterative methods.

5.1 - Asynchronous iteration with memory

A generalization to definition 1 can be obtained by noting that, if, for $j = 2, 3, \ldots$, it happens that $k_j = j-2$ in equation (5.1), this equation defines a sequence of iterates which corresponds exactly to the sequence generated by an iterative method with one memory. This remark suggests the following generalization for the problem stated in equation (1.1).

Given an operator F from $[\mathbb{R}^n]^m$ into $\mathbb{R}^n$, the problem is now to find a vector ξ in $\mathbb{R}^n$ such that:

$$\xi = \lim_{\{x^1 \to \xi, \ldots, x^m \to \xi\}} F(x^1, \ldots, x^m). \tag{5.2}$$

The vector ξ will still be called a *fixed point* for the operator F.

In very much the same way as we introduced the class of asynchronous iterative methods to solve equation (1.1), we now introduce the class of ***asynchronous iterative methods with memory*** to solve equation (5.2).

Definition 4:

Let F be an operator from $[\mathbb{R}^n]^m$ into $\mathbb{R}^n$. An ***asynchronous iteration with***

memory corresponding to the operator F and starting with a given set of vectors $x(0), \ldots, x(m-1)$ is a sequence $x(j)$, $j = 0, 1, \ldots$, of vectors of $\mathbb{R}^n$ defined for $j = m, m+1, \ldots$ by:

$$x_i(j) = \begin{cases} x_i(j-1) & \text{if } i \notin J_j \\ f_i(z^1, \ldots, z^m) & \text{if } i \in J_j, \end{cases}$$

where z^r, $1 \le r \le m$, is the vector with components $z_i^r = x_i(s_i^r(j))$, $1 \le i \le n$. As in definition 1, $\mathcal{J} = \{ J_j \mid j = m, m+1, \ldots \}$ is a sequence of non-empty subsets of $\{1, \ldots, n\}$ which correspond to the subsets of components evaluated at each step of the iteration. But the sequence $\mathcal{S}$ is now to be replaced by:

$$\mathcal{S} = \{ (s_1^1(j), \ldots, s_n^1(j), s_1^2(j), \ldots, s_n^m(j)) \mid j = m, m+1, \ldots \},$$

a sequence of elements in $[\mathbb{N}^n]^m$. In addition, while condition (c) of definition 1 remains the same, conditions (a) and (c) now become:

for each $i = 1, \ldots, n$

(a) $\max\{ s_i^r(j) \mid 1 \le r \le m \} \le j-1$, for $j = m, m+1, \ldots$,

(b) $\min\{ s_i^r(j) \mid 1 \le r \le m \} \le j-1$ tends to infinity as j tends to infinity.

An asynchronous iteration with memory corresponding to F, starting with a set X of m vectors and defined with $\mathcal{J}$ and $\mathcal{S}$ will be denoted by $(F, X, \mathcal{J}, \mathcal{S})$. ∎

For practical reasons (e. g., stability in the implementation on a computer), we might want to have the additional condition that the vectors $z^1, \ldots, z^m$ are all distinct. But this restriction is not essential for our purpose here if we assume, for example, that the operator F is defined by continuity when two or more vectors are identical. This will be the case with the class of operators we will consider.

Now, in order to obtain, for asynchronous iterations with memory, a convergence result similar to the result stated in theorem 1, we need to generalize the notion of contracting operators to operators from $[\mathbb{R}^n]^m$ into $\mathbb{R}^n$.

In the remainder of the section, we will use the following notation. If $\{x^1, \ldots, x^m\}$ is a set of vectors in $\mathbb{R}^n$, $z = \max[x^1, \ldots, x^m]$ denotes the vector in $\mathbb{R}^n$ with components $z_i = \max\{ x_i^r \mid 1 \le r \le m \}$. A natural generalization to the notion of contracting operators is given in the following.

Definition 5:

An operator F from $[\mathbb{R}^n]^m$ into $\mathbb{R}^n$ is an *m-contracting operator* on a subset D

of $\mathbb{R}^n$ if there exists a non-negative $n \times n$ matrix A with spectral radius less than unity satisfying, for all $x^1, \ldots, x^m, y^1, \ldots, y^m$ in D,

$$|F(x^1, \ldots, x^m) - F(y^1, \ldots, y^m)| \leq A \max[|x^1 - y^1|, \ldots, |x^m - y^m|] .$$

The matrix A will be called a contracting matrix for the operator F. ∎

When $m = 1$, the preceding definition corresponds exactly to definition 3. And m-contracting operators have all the properties we have already mentioned for contracting operators. In particular, it is clear from the definition that m-contracting operators are continuous and, in fact, uniformly continuous on D^m. The uniqueness of a fixed point in D is also easily derived. In addition, if we assume that D is a closed subset of $\mathbb{R}^n$ satisfying $F(D^m) \subset D$, then we are guaranteed of the existence of a fixed point in D: the fixed point is, for example, obtained as the limit of the sequence $x(j)$, $j = 0, 1, \ldots$, defined by:

$$x(j) = F(x(j-1), \ldots, x(j-m)) , \quad j = m, m+1, \ldots ,$$

which is independent of the set of starting vectors $x(0), \ldots, x(m-1)$ in D.

We are now able to state the analogue of theorem 1 for m-contracting operators in the following.

Theorem 2:

If F is an m-contracting operator on a closed subset D of $\mathbb{R}^n$ satisfying $F(D^m) \subset D$, then any asynchronous iteration with memory corresponding to the operator F and starting with an arbitrary set of m vectors in D converges to the unique fixed point of F in D.

Proof:

With slight modifications, the proof of this theorem is identical to the proof of theorem 1. ∎

5.2 - Examples of asynchronous iterations with memory

In the beginning of this section, we considered the *Asynchronous Newton's method* to find the simple root ξ of a non-linear operator G. This method led to the sequence of iterates generated by the asynchronous iteration with memory $(H, \{x(0), x(0)\}, \mathcal{J}, \mathcal{S})$, where:

$$J_j = \{1, \ldots, n\} \quad \text{for } j = 2, 3, \ldots ,$$

$$s_i^1(j) = j-1 , \; s_i^2(j) = k_j \quad \text{for } j = 2, 3, \ldots \text{ and } i = 1, \ldots, n .$$

In addition, as the operator H can easily be shown to be a 2-contracting operator (assuming, for example, some Lipchitz condition for the derivative of G in a small neighbor of the root ξ), we see that the sequence defined by equation (5.1) converges to ξ, provided that k_j tends to infinity with j (which simply states the fact that the processes eventually complete each step of their computations).

Let F be an operator from $[\mathbb{R}^n]^m$ into $\mathbb{R}^n$, and let ω be a positive scalar. Consider the operator F_ω from $[\mathbb{R}^n]^{m+1}$ into $\mathbb{R}^n$ obtained from the operator F by the introduction of the relaxation factor ω, and defined as

$$F(x^0, x^1, ..., x^m) = (1-\omega)x^0 + \omega F(x^1, ..., x^m).$$

We first note that both F and F_ω have the same fixed points (if any). We also note that, if F is an m-contracting operator on some subset D of $\mathbb{R}^n$ with the contracting matrix A, then, for all $x^0, x^1, ..., x^m, y^0, y^1, ..., y^m$ in D, the operator F_ω satisfies:

$$\begin{aligned} |F_\omega(x^0, ..., x^m) - F_\omega(y^0, ..., y^m)| &\le |1-\omega||x^0-y^0| + \omega|F(x^1, ..., x^m)-F(y^1, ..., y^m)| \\ &\le |1-\omega||x^0-y^0| + \omega A \max[|x^1-y^1|, ..., |x^m-y^m|] \\ &\le [|1-\omega|I + \omega A]\max[|x^0-y^0|, |x^1-y^1|, ..., |x^m-y^m|], \end{aligned}$$

and, provided that $0 < \omega < 2/[1+\rho(A)]$, F_ω is an $(m+1)$-contracting operator on D with the contracting matrix $A_\omega = |1-\omega|I + \omega A$. This reestablishes, in a more general setting, the result mentioned in section 3.2 for asynchronous iterative methods with relaxation.

Many more examples of asynchronous iterations with memory can be given and, in particular, all classical iterative method with memory can be expressed in this way. In addition, all usual super-linear iterative methods with m memories can be shown (under weak conditions) to correspond to some $(m+1)$-contracting operator, therefore ensuring the convergence of any asynchronous iterations corresponding to this operator.

6 - On the complexity of asynchronous iterations

Let F be an operator from $\mathbb{R}^n$ to itself with a fixed-point ξ and satisfying the assumptions of theorem 1. We now investigate some measures of efficiency for the convergence of the asynchronous iteration $(F, x(0), \mathcal{J}, \mathcal{S})$ toward the fixed-point ξ of F.

The constructive proof of the theorem already provides us with bounds for the error vector $x(j) - \xi$. And, in fact, if F is a contracting operator with the contracting matrix A, we note that an estimate of the error committed with the asynchronous iteration $(F, x(0), \mathcal{J}, \mathcal{S})$ is directly obtainable from the asynchronous iteration $(A, |x(0)-\xi|, \mathcal{J}, \mathcal{S})$. This

estimate is used in this section to derive bounds for the efficiency of asynchronous iterations corresponding to contracting operators. However, since $(A,|x(0)-\xi|,\mathcal{J},\delta)$ can only reflect linear convergence, this estimate is certainly not adequate to deal with all asynchronous iterations, and, in section 8, using an example, we present an analysis for an asynchronous iteration with super-linear convergence.

For convenience, we only consider the convergence in norm of the error vector $x(j) - \xi$. By choosing, for example, the norm $\|x\| = \max\{ |x_i| \mid i = 1, ..., n \}$, this corresponds to the worst possible case for the convergence of the components.

To measure the linear convergence of the sequence $x(j)$, $j = 0, 1, ...$, toward its limit ξ, we consider the following complexity measures often referred to in the literature. The rate of convergence of the sequence is defined as:

$$\mathcal{R} = \liminf_{j\to\infty} [(-\log\|x(j)-\xi\|)/j] .$$

In addition, if c_j is the cost associated with the evaluations of the first j iterates, $x(1), ..., x(j)$, we define the efficiency of the sequence by:

$$E = \liminf_{j\to\infty} [(-\log\|x(j)-\xi\|)/c_j] .$$

If all logarithms are taken to the base 10, $1/\mathcal{R}$ measures the asymptotic number of steps required to divide the error by a factor of 10, whereas $1/E$ measures the corresponding cost. We note that, if c_j/j tends to some finite limit c (which corresponds to the average cost per step), then the efficiency is simply given by $E = \mathcal{R}/c$.

The costs c_j, $j = 1, 2, ...$, can be chosen according to any convenient measure. In our case, we consider the cost to correspond either to the number of evaluations of the operator F, or to the time to perform the evaluations. In the former case, if each component is equally as hard to compute, the cost can be directly evaluated from the sequence $\mathcal{J}$ by considering

$$c_j = (|J_1| + ... + |J_j|)/n , \tag{6.1}$$

where $|J_j|$ is the cardinality of the set J_j, i. e., the number of components evaluated at the j-th step of the iteration. In the latter case, the cost is better suited to deal with parallel algorithms, and can be evaluated through the classical tools of queueing theory. When it is necessary to indicate which cost measure is used in the evaluation of the efficiency, we use the notations E_e if the cost is measured in number of evaluations of F, and E_t if the cost is measured by the time needed to perform (sequentially) one evaluation of F.

6.1 - General bounds

We return to the proof of theorem 1, and we use the same notations. The proof simply consists of constructing an increasing sequence of indices j_p, $p = 0, 1, \ldots$, satisfying

$$\|x(j) - \xi\| \leq \alpha\omega^p \quad \text{for} \quad j \geq j_p ,$$

where the positive constant α can be taken to be $\alpha = \|x(j)-\xi\|$. From the construction of this sequence we note that

$$j_{p+1} = j_p + r_p + t_p \quad \text{for} \quad p = 0, 1, \ldots ,$$

where r_p and t_p are integers chosen to satisfy: (1) starting with the index j_p+r_p, all evaluations of iterates do not make any more use of values of components corresponding to iterates with indices smaller than j_p; and (2) all components are evaluated at least once between the (j_p+r_p)-th and the $(j_p+r_p+t_p)$-th iterates.

Now let

$$p_j = \sup\{ p \mid r_0+t_0+\ldots+r_{p-1}+t_{p-1} \leq j \} \quad \text{for} \quad j = 0, 1, \ldots . \tag{6.2}$$

Then, if we know r_p and t_p for $p = 0, 1, \ldots$, we can deduce a bound on $\|x(j)-\xi\|$ since

$$\|x(j)-\xi\| \leq \alpha\omega^{p_j} \quad \text{for} \quad j = 0, 1, \ldots ,$$

which shows that the sequence $x(j)$, $j = 0, 1, \ldots$, converges at least as fast as the sequence ω^{p_j}, $j = 0, 1, \ldots$, with a rate of convergence $\mathcal{R}$ such that

$$\mathcal{R} \geq - [\liminf_{j\to\infty} (p_j/j)] \log\omega .$$

And, if c_j is the cost associated with the evaluations of the first j iterates, we have the following bound for the efficiency:

$$E \geq - [\liminf_{j\to\infty} (p_j/c_j)] \log\omega .$$

In addition, as was noticed earlier, if A is a contracting matrix for the operator F, ω can be chosen arbitrarily close to $\rho(A)$. This shows that in the bounds we have just obtained we can simply replace ω by $\rho(A)$, and this yields the following.

Theorem 3:

Let F satisfy the condition of theorem 1, and let A be a contracting matrix for the operator F. Then the asynchronous iteration $(F,x(0),\mathcal{J},\mathcal{S})$ converges to the fixed point of F with a rate of convergence

$$\mathcal{R} \geq - [\liminf_{j\to\infty} (p_j/j)] \log\rho(A) ,$$

and an efficiency

$$E \geq - [\liminf_{j\to\infty} (p_j/c_j)] \log\rho(A) ,$$

where the sequence p_j is defined from $\mathcal{J}$ and $\mathcal{S}$ by equation (6.2).

An example

As an illustration, we consider the parallel implementation of Jacobi's method with k processes. For simplicity, we assume that n is a multiple of k, and we set $q = n/k$.

To avoid an overhead in the selection of the components to be updated at each step of the iteration, each process is assigned to the evaluation of a fixed subset of the components. In particular, when all components are equally as hard to compute, and when all processors are equally as fast, it is natural to decompose the set of components into subsets of equal sizes, and, for example, to assign the first process to the evaluation of the first q components, the second process to the evaluation of the next q components, and so forth. Corresponding to this decomposition, a parallel implementation of Jacobi's method with k processes can be represented by the asynchronous iteration $(F, x(0), \mathcal{J}, \mathcal{S})$, where $\mathcal{J}$ and $\mathcal{S}$ are defined by:

$$J_j = \{ i \mid 1 + (j-1 \bmod k)q \le i \le q + (j-1 \bmod k)q \} \quad \text{for} \quad j = 1, 2, \ldots ,$$

$$s_i(j) = \lfloor (j-1)/k \rfloor q \quad \text{for} \quad j = 1, 2, \ldots \text{ and } i = 1, \ldots, n .$$

The two asynchronous iterations we introduced in section 2.2 to represent Jacobi's method correspond to the particular cases $k = 1$ and $k = n$.

It is easy to check that r_p and t_p are given by 1 and k, respectively, for $p = 0, 1, \ldots$. This shows that $p_j = \lfloor j/k \rfloor$ and therefore

$$\mathcal{R}(k) \ge -(\log \rho(A))/k .$$

Now, if c_j measures the number of evaluations of F required to compute the first j iterates, using equation (6.1), we have $c_j = j/k$. This gives for the efficiency:

$$E_\infty(k) \ge -(\log \rho(A)) . \tag{6.3}$$

For all values of k, we obtain the same bound for the efficiency. In particular, when F is the linear operator defined by $F(x) = Ax + b$, where A is a non-negative $n \times n$ matrix with spectral radius less than unity, then A can be chosen as a contracting matrix for F and the bound (6.3) is known to be sharp.

Since the asynchronous iteration we are considering corresponds to a parallel implementation of Jacobi's method, instead of measuring the cost by the number of evaluations of F, it is more natural to use the average time to perform the evaluations as a measure of the cost. Let the time unit be the average time to perform (sequentially) one evaluation of F. Then, if $pk \le j \le (p+1)k$, we have $c_{pk} \le c_j \le c_{(p+1)k}$ and $c_{pk} = p[\lambda_k/k]$. The expression λ_k/k corresponds to the time for the k processes to

execute in parallel their computations and to synchronize their executions. The factor λ_k is the *penalty factor* mentioned in [3]; it measures the overhead due to the fluctuations in the computing times of the k processes, and can be evaluated if we know, for example, the distribution function for the time to evaluate F. In particular, we have $\lambda_1 = 1$ and, for $k \geq 2$, $\lambda_k \geq 1$ with the equality only when it always take the same constant time to evaluate F (i. e., there are no fluctuations in the computing time). This cost measure yields the following bound for the efficiency:

$$E_t(k) \geq -[k/\lambda_k]\log\rho(A) .$$

Again, these bounds are sharp for the linear operator we mentioned above, and the ratio $E_t(k)/E_t(1) = k/\lambda_k$ measures the speed-up achieved by using a parallel implementation with k processes. We would expect the implementation with k processes to be k times as efficient as the sequential implementation (with $k = 1$), but this is not so because of the overhead introduced by synchronizing the k processes and measured by the penalty factor λ_k.

6.2 - Additional assumptions

In the preceding example, we have been able to carry out the analysis for Jacobi's method (and even obtain sharp bounds on the efficiency) because the representation in terms of asynchronous iterations is known explicitly and follows a very regular pattern. This is not, however, generally so. For example, in a parallel implementation with several processes using no synchronization (as presented in section 2.1), the sequences $\mathcal{S}$ and $\mathcal{J}$ (and, therefore, the sequences r_p and t_p, $p = 0, 1, \ldots$) are not known directly but are only defined implicitly by the processes in the course of their executions.

Below, we present alternate bounds for $\mathcal{R}$ and E under conditions often satisfied in usual implementations of asynchronous iterations. We assume that we know bounds on r_p and t_p, and we restrict the definition of the class of asynchronous iterative methods by replacing conditions (b) and (c) of definition 1 with the following:

(b') There exists a positive integer r such that, for $j = 1, 2, \ldots$ and $i = 1, \ldots, n$, $s_i(j) \geq j-r$,

(c') there exists a non-negative integer t such that, for $j = 1, 2, \ldots$, $J_j \cup \ldots \cup J_{j+t} = \{1, \ldots, n\}$.

Condition (b') corresponds exactly to the restriction stated by Chazan and Miranker in the definition of the chaotic relaxation scheme [1]. We have criticized the condition for the

generality of the definition, and we have shown that this condition was not necessary to ensure the convergence of asynchronous iterations. In practical applications, however, this condition is often satisfied, in particular, when the computations of all components have the same complexity (which is the case with a linear operator). Condition (c') is also satisfied for most of the usual implementations of asynchronous iterations, since it is natural that (1) a process evaluates a component by using the most recently updated values of all components; and (2) two processes never evaluate the same component at the same time; in this case it follows directly that, by taking $r = t+1$, conditions (b') and (c') are equivalent.

Under the additional conditions (b') and (c'), we clearly have $r_p \leq r$ and $t_p \leq t$, for $p = 0, 1, \ldots$, and, therefore, $p_j \leq \lfloor j/(r+t) \rfloor$. From the bounds stated in theorem 3, we immediately obtain the following.

Corollary:

Let F satisfy the condition of theorem 1, and let A be a contracting matrix for F. If the asynchronous iteration $(F, x(0), \mathcal{J}, \mathcal{S})$ satisfies the additional conditions (b') and (c'), then it converges to the fixed point of F with a rate of convergence

$$\mathcal{R} \geq - [1/(r+t)] \log \rho(A),$$

and an efficiency

$$E \geq - [\lim_{j \to \infty} j/(r+t)c_j] \log \rho(A),$$

where the sequence p_j is defined from $\mathcal{J}$ and $\mathcal{S}$ by equation (6.2).

7 - Experimental results

Several asynchronous iterations have been experimented with on C.mmp, the Carnegie-Mellon multiprocessor [8], and the actual measurements are presented in the next section. The different asynchronous iterative methods are described below.

7.1 - Asynchronous iterations experimented with

All asynchronous iterations we have experimented with consist of the parallel execution of k processes. As we did with the parallel implementation of Jacobi's method, we assign to each of the processes the evaluation of a fixed subset of the components. Each process computes cyclically new values for the components in its subset, and the methods only differ by the choices of the values used in the evaluations.

Asynchronous Jacobi's method (AJ): For the evaluations of all components, a process uses only values of the components known at the beginning of a *cycle*, and the process releases all new values at the end of each cycle.

Asynchronous Gauss-Seidel's method (AGS): Same as the AJ method except that the process uses new values of the components in its subset as soon as they are known for further evaluations in the same cycle. Again, it releases the new values (for the other processes) at the end of its cycle.

Purely Asynchronous method (PA): A process computes the new values of each component by using the most recent values of all components and releases each new value immediately after its evaluation.

The PA method is certainly the easiest method to implement, and, as far as space is concerned, is clearly the most efficient one, whereas the AJ method is the worst one, since it requires from each process not only a complete duplication of all components (as of the beginning of its cycle) but still another copy of the components in its own subset. This can hardly be justified but experimental results give useful comparisons between the AJ method and the actual Jacobi's method (also between the AGS and Gauss-Seidel's methods).

In addition, both the AJ and AGS methods also require the need for a critical section in order to read all components at the beginning of a cycle and to update the values at the end of a cycle, whereas no critical section is needed with the PA method. However, C.mmp has the drawback that no indivisible instructions exist to read or write floating point numbers (implemented on two consecutive words of memory), therefore, if we are to implement the PA method on C.mmp, only the first 8 bits of the mantissa can be considered significant, and the admissible error in the termination criterion has to be chosen accordingly.

7.2 - Results

The three methods just described, as well as Jacobi's method, have been implemented on C.mmp to solve the Dirichlet problem for Laplace's equation on a rectangular domain of $\mathbb{R}^2$. Using the method of finite differences, an approximate solution to this problem can be found by solving a linear system of equations. In the experiments reported here, a regular grid has been chosen with 21×24 interior points,

resulting in a linear system of size $n = 504$. This system can be represented in the form $x = F(x) = Ax + b$, where the vector b is obtained from the boundary conditions, and the matrix A is a (very sparse) non-negative matrix with spectral radius $\rho(A) \doteq 0.991$. Since $\rho(|A|) = \rho(A) < 1$, this shows that A is a contracting matrix for the operator F, and, therefore, that the result of theorem 1 can be applied to F to ensure the convergence of each iterative method.

At the time the measurements have been taken, the configuration of C.mmp included six processors, and all iterative methods have been run with a number of processes $k = 1, 2, 3, 4$, and 6. Each of the results reported here is the average of three measurements, but, since C.mmp was used in stand-alone during the experiments, very little difference was noted from one run to the next.

In table 1, we report for the four methods the average number of vector evaluations required to reduce (asymptotically) the error vector by a factor of 10: this corresponds to the cost measure $1/E_e$. And, in table 2, we report the average time (expressed in seconds) required to achieve this reduction: this corresponds to the cost mesure $1/E_t$.

The bounds obtained from the results of the previous sections are mentioned in parentheses along with the measurements. The parameters in these bounds have been evaluated either directly (e. g., $\rho(A) \doteq 0.991$), or through measurements by tracing the executions of the processes. In particular, for the AJ, AGS and PA methods, the bounds r and t, defined in section 6.2, have been determined by observing the sequencing of the tasks performed by the different processes. Similarly, the penalty factor in Jacobi's method and the overhead due to the critical section in the AJ and AGS methods have been obtained by direct measurements: they are presented in tables 3 and 4.

	Jacobi	AJ	AGS	PA
k = 1	254 (254)	254 (254)	127 (254)	127 (254)
k = 2	254 (254)	266 (888)	142 (888)	127 (762)
k = 3	254 (254)	267 (846)	149 (846)	127 (762)
k = 4	254 (254)	273 (825)	166 (825)	129 (762)
k = 6	254 (254)	285 (804)	196 (804)	128 (762)

Table 1 - Number of evaluations required to divide the error by a factor of 10

	Jacobi	AJ	AGS	PA
k = 1	337 (337)	337 (337)	168 (337)	168 (337)
k = 2	241 (241)	211 (705)	113 (705)	84 (506)
k = 3	178 (178)	149 (471)	83 (471)	56 (337)
k = 4	153 (153)	123 (372)	75 (372)	43 (253)
k = 6	131 (131)	102 (289)	70 (289)	28 (169)

Table 2 - Time required to divide the error by a factor of 10

	k = 1	k = 2	k = 3	k = 4	k = 6
λ_k	1	1.43	1.59	1.82	2.34
%	0	29.9	37.1	45.1	57.3

Table 3 - Penalty factor with Jacobi's method and percentage of the wasted time

	k = 1	k = 2	k = 3	k = 4	k = 6
λ_k	1	1.20	1.26	1.35	1.62
%	0	16.6	20.8	26.0	38.2

Table 4 - Critical section overhead cost with the AJ and AGS methods and percentage of the wasted time

These results must only be considered to illustrate the behavior of asynchronous iterations, since, in particular, the two cost measures reported in tables 1 and 2 strongly depend on both the problem (i. e., the matrix A) and the multiprocessor system. Yet, they show a clear advantage of asynchronous methods over synchronized methods.

We note, for example, from table 3 that, with Jacobi's method, when $k = 6$ processes are used, the penalty factor is as big as $\lambda_6 = 2.34$. This means that about 57 percent of the time is spent by a process waiting for the other processes to finish their computations. This limits the possible speed-up to 2.6 rather than 6.

We also note that the use of critical sections, too, should be avoided, since, with the AJ or AGS methods, when 6 processes are used, about 38 percent of the time is spent waiting for entering the critical section, again limiting the possible speed-up to 3.7 rather than 6.

The measurements for the PA method, on the other hand, indicate that we achieve an almost full speed-up with this method (at least with a small number of processes). An obvious reason for this speed-up is the total absence of any form of synchronization; another reason, specific to the problem we have experimented with and indicated by the results of table 1, is the sparsity of the matrix A.

The bounds derived in section 6 have been obtained in a very general case. Yet tables 1 and 2 show that they are always within a factor between 3 and 6 of the actual measurements (except for Jacobi's method where they are sharp). In addition, we certainly could obtain much sharper bounds by carrying out the analysis for the specific problem we have experimented with (for example, by taking into account the sparsity of the matrix). In particular, a specific analysis for the PA method can easily explain the fact that $1/E_e$ is almost not influenced by the number of processes (see table 1).

8 - Asynchronous iterations with super-linear convergence

As we already noticed, the bounds established in section 6 are certainly not adequate to measure the complexity of iterations with super-linear convergence. In this section, we use as an example the iterative method we have mentioned at the beginning of section 5 to show how an analysis of the complexity can be done for this case.

To study the convergence of a sequence $x(j)$, $j = 0, 1, \ldots$, toward its limit ξ, we now use the following usual measures of complexity. The order of convergence is defined as

$$\rho = \liminf_{j\to\infty} [(-\log\|x(j)-\xi\|)^{1/j}],$$

and, as before, if c_j is the cost associated with the evaluations of the first j iterates, $x(1), \ldots, x(j)$, we define the efficiency of the sequence by:

$$E = \liminf_{j\to\infty} [(\log-\log\|x(j)-\xi\|)/c_j],$$

Again, we note that, if the average cost per step c_j/j tends to some finite limit c when j tends to infinity, the efficiency is simply given by $E = (\log\rho)/c$. In the remainder of the section, we assume that the limit c exists.

In order to find the simple root ξ of an operator G from $\mathbb{R}^n$ into itself, we use the *Asynchronous Newton's method*, AN, as implemented by the two processes described at the beginning of section 5. Let r_i, $i = 1, 2, \ldots$, be the number of iterates evaluated by the first process, *P1*, during the i-th evaluation of the derivative G' by the second process, *P2*. Let $j_0 = 0$ and $j_i = r_1 + \ldots + r_i$ for $i = 1, 2, \ldots$, then $x(j_i)$, $i = 0, 1, \ldots$, is the iterate used

by *P2* for the *(i+1)*-st evaluation of the derivative. Starting with the two initial values $x(0)$ and $G'(x(0))$, the AN method generates with the two processes *P1* and *P2* the sequence of iterates $x(j)$, $j = 1, 2, \ldots$, defined by

$$x(j+1) = x(j) - [G'(x(j_{i-1}))]^{-1}G(x(j)) \text{ , for } i = 1, 2, \ldots \text{ and } j_i < j \le j_{i+1} . \quad (8.1)$$

The following theorem gives the measures of complexity for this sequence if we know some bounds on the sequence r_i, $i = 1, 2, \ldots$.

Theorem 4:

Let the initial approximation $x(0)$ be close enough to the root ξ, that is

$$x(0) \in D_\varepsilon = \{ x \mid \|x-\xi\| < \varepsilon \} ,$$

and let the derivative G' satisfy some Lipchitz condition on D_ε:

$$\|G'(x)-G'(y)\| \le M\|x-y\| , \quad \forall\, x, y \in D_\varepsilon .$$

If ε satisfies the condition

$$M\|G'(\xi)^{-1}\|\varepsilon < 2/5 ,$$

and if there exist some positive integers p and q such that

$$p \le r_i \le q , \text{ for } i = 1, 2, \ldots ,$$

then the order of convergence, ρ, and the efficiency, E, of the sequence defined by equation (8.1) satisfy:

$$\rho \ge \lambda_p^{1/q} , \quad (8.2)$$

and

$$E \ge (\log\lambda_p)/(q\varepsilon) , \quad (8.3)$$

where λ_p is the largest root of the equation $z^3 - z^2 - (p-1)z - 1 = 0$ (for which we can check easily that $0.4 + \sqrt{p} < \lambda_p < 0.5 + \sqrt{p}$, $p = 1, 2, \ldots$).

Proof:

The proof is easy but technical, and below we only give an outline for this proof.

Let $\alpha = M\|G'(\xi)^{-1}\|$, and let $c = 3\alpha/[2(1-\alpha\varepsilon)]$. From the choice of ε, we first note that, starting with $x(0) \in D_\varepsilon$, the sequence $\|x(j)-\xi\|$, $j = 0, 1, \ldots$, is strictly decreasing and satisfies:

$$\|x(j_i+1)\xi\| \le c\|x(j_{i-2})-\xi\|\|x(j_i)-\xi\| , \text{ for } i = 2, 3, \ldots ,$$

and

$$\|x(j+1)\xi\| \le c\|x(j_{i-1})-\xi\|\|x(j)-\xi\| , \text{ for } i = 2, 3, \ldots \text{ and } j_i < j < j_{i+1} = j_i + r_i .$$

By substitution, it follows that, for $i = 2, 3, \ldots$,

$$\|x(j_{i+1})-\xi\| \le c^{r_i}\|x(j_{i-1})-\xi\|^{r_i-1}\|x(j_{i-2})-\xi\|\|x(j_i)-\xi\| ,$$

and, if we set $u_i = -\log c\|x(j_i)-\xi\|$, we obtain:

$$u_{i+1} \geq u_i + (r_i - 1)u_{i-1} + u_{i-2}, \quad \text{for } i = 2, 3, \ldots .$$

Therefore, by using the lower bound on r_i, we deduce that

$$u_{i+1} \geq u_i + (p-1)u_{i-1} + u_{i-2}, \quad \text{for } i = 2, 3, \ldots .$$

This shows that u_i tends to infinity at least as fast as λ_p^i. Therefore, the order of convergence, ρ', of the subsequence $x(j_i)$, $i = 0, 1, \ldots$, must verify $\rho' \geq \lambda_p$. The bounds (8.2) and (8.3) are derived directly from this last inequality. ∎

In particular, if the cost c_j measures the number of evaluations of the operator G, we simply have $c_j = j$, and, therefore, $E_e \geq (\log\lambda_p)/q$. On the other hand, if the cost corresponds to the execution time, the efficiency will depend on the implementation itself. For example, an implementation corresponding strictly to the generation of the sequence described by equation (8.1) requires the use of a critical section for reading and writing, in a block, the values of the iterates and of the derivative. The use of a critical section introduces an overhead, but, as is done with the PA method, the overhead can be avoided if a process uses whatever values are currently available when needed. In this case the bounds of theorem 4 still holds, and ε can be given the value $\varepsilon = 1$.

The parameters p and q, too, depend on the particular implementation of the AN method, and, especially, on the relative speeds of the processors executing the processes *P1* and *P2*. In practice, if the processors are equally as fast, we expect, with small variations, r_i to be close to n, and the values $p = q = n$ can predict good estimates for the efficiency of the AN method implemented with two processes.

The AN method is easily generalizable to more than two processes. If k processes are available, k_1 might be assigned to the evaluation of the sequence of iterates, while $k_2 = k - k_1$ are assigned to the evaluation of the derivative. The bounds of theorem 4 still holds for this case as well, only with different values for the sequence r_i, $i = 1, 2, \ldots$ (or for the bounds p and q), determined by the parallel implementations of the two evaluations. Further results in this direction will be reported elsewhere.

9 - Extensions of the results

We mention below some direct extensions of the results presented in this paper and some points subject to further developments.

A straighforward generalization of the results can be obtained if, instead of $\mathbb{R}^n$, we

consider the product P of n Banach spaces B_i with norms $|.|_i$, $i = 1, \ldots n$. In this case, if x is an element of P, x is determined by its components $x_i \in B_i$, $i = 1, \ldots n$. And $|x|$ represents the non-negative vector of $\mathbb{R}^n$ with components $|x_i|_i$, $i = 1, \ldots n$.

Considering only the class of linear operators, $F(x) = Ax + b$, we have noted that the notion of contracting operators coincides with the condition that $\rho(|A|) < 1$. In [1], Chazan and Miranker have shown that this condition is not only sufficient but also necessary for the convergence of all chaotic iterations. This implies, in particular, that all asynchronous iterations corresponding to a linear operator F are convergent if and only if F is a contracting operator. When we also consider non-linear operators, however, the proof given by Chazan and Miranker does not apply any more, and it would be of interest to obtain conditions on the class of operators for which all asynchronous iterations are guaranteed to converge. Similar conditions for the convergence of a more restricted class of iterations would also be of interest, in particular, for the subclass of asynchronous iterative methods corresponding to the additional assumptions introduced in section 6.1.

The bounds we have obtained to estimate the rate of convergence of asynchronous iterations have been derived by considering the worst possible case, and, compared to actual measurements, these bounds happen to be very conservative. It would certainly be very useful to obtain bounds (or estimates) corresponding to the average behavior of asynchronous iterations, for example, given the probability distributions of the two sequences $\mathcal{J}$ and $\mathcal{S}$, or, more generally, given the distribution functions for the time it takes the different processes to evaluate the components.

We have already mentioned the possibity to introduce a relaxation factor in asynchronous iterations, and, for contracting operators, we have derived a possible range that guarantees the convergence of all asynchronous iterations. Nothing is known, however, about the *optimal* choice of the relaxation factor, for example, given directly the asynchronous iteration through $\mathcal{J}$ and $\mathcal{S}$, or, again, given the distribution functions for the evaluation times.

10 - Concluding remarks

In the implementation of most parallel algorithms, synchronization seems to be required to assure the communication between the processes, and to guarantee their

correct executions. However, the main drawback with synchronization is that it degrades considerably the performance of the algorithms because it is very time consuming. The class of *asynchronous iterative methods* avoids this drawback. It includes iterations corresponding to a parallel implementation in which the cooperating processes have a minimum of intercommunication and do not make any use of synchronization. The *Purely Asynchronous method* described in section 7.1 is a typical example of an asynchronous iterative method.

In [1], Chazan and Miranker introduced chaotic relaxation schemes requiring a condition which can only be satisfied by using repeated checking and some form of synchronization at each step of the iteration. Asynchronous iterative methods do not require this condition and are more general than chaotic relaxation schemes. Asynchronous iterations further generalize to *asynchronous iterations with memory* which allow different values of the same variable to be used within the same computation.

Using the notions of *contracting operators* and of *m-contracting operators*, theorem 1 and theorem 2 state sufficient conditions to guarantee the convergence of any asynchronous iterations and asynchronous iterations with memory. These conditions are satisfied for a large class of operators.

In the second part of the paper, asynchronous iterations are evaluated from a computational point of view, then the results of a series of actual measurements (obtained by running asynchronous iterations on a multiprocessor) are presented. These results fully justify the use of asynchronous iterative methods.

General bounds on the efficiency of asynchronous iterations are first derived directly from the proof of the convergence theorem. Although these bounds are sharp for a parallel implementation of Jacobi's method, they are of little applicability since they require to know *a priori* the exact specification of each step of the iteration. Alternate bounds are then derived under additional conditions which are usually satisfied in practical applications. These bounds are consistent with actual measurements; for the experiments we have run, they are always within a factor of 6 of the measurements. In addition, it is our feeling that these bounds can be largely improved if we take into account specific characteristics of the problem being solved, therefore leading to a better understanding of asynchronous iterations. In section 8, for example, we have made a first step in this direction, and we have presented an analysis for the *Asynchronous Newton's method*.

A series of experiments has been conducted on C.mmp, a multiprocessor system (with 6 processors at the time the experiments have been run), and several asynchronous iterative methods have been implemented to solve a large linear system of equations. They range from Jacobi's method, requiring a full synchronization of all the processes at each step of the iteration, to the PA method, which requires no synchronization at all. In between, the AJ and AGS methods are derived from the usual Jacobi's and Gauss-Seidel's methods, and they require the use of a critical section.

The experimental results show a considerable advantage for the iterative method with no synchronization at all. For a number of processes up to the number of processors available on C.mmp, the PA method exhibits full parallelism and has an optimal speed-up compared to Gauss-Seidel's method, the best sequential method experimented with. The AJ and AGS methods have a very similar behavior, and when 6 processes are used the overhead caused by the critical section implies that 38 percent of the time a process is waiting for entering the critical section. As is intuitively expected, Jacobi's method has the worst behavior of all the methods considered, and, with 6 processes, the overhead, due to the synchronization of all the processes at each step of the iteration, is about 57 percent (i. e., more than half the time a process is waiting for the other processes to finish their computations).

On the basis of these experimental results, and for the problem we have considered, there does not seem to be any alternatives: the PA method is obviously the most efficient one. In addition, another advantage of the PA method is that it is the easiest one to implement, and, spacewise, it is also the most efficient one.

Finally, another possibility, which has only been outlined in the paper, is the introduction of a relaxation factor. Based only on a few experimental results (not reported here), it is our belief that we can expect an improvement of the *Purely Asynchronous Over-Relaxation method* over the PA method similar to the improvement of the SOR method over the Gauss-Seidel's method, if we choose the relaxation factor in an optimal way. The optimal choice of the relaxation factor depends not only on the system being solved, but also on the probability distributions of the various execution times by the different processes.

Acknowledgements

I wish to thank H. T. Kung and J. F. Traub, at Carnegie-Mellon University, and H. Woźniakowski, at the University of Warsaw, for reading and commenting on this paper. I am especially grateful to H. T. Kung, whose extensive suggestions have significantly shaped the final version.

Bibliography

1. D. Chazan and W. Miranker, Chaotic relaxation, *Linear Algebra and Its Applications*, Vol. 2, 1969, pp. 199-222.

2. J. D. P. Donnelly, Periodic chaotic relaxation, *Linear Algebra and Its Applications*, Vol. 4, 1971, pp. 117-128.

3. H. T. Kung, Synchronized and asynchronous parallel algorithms for multiprocessors, Carnegie-Mellon University, Computer Science Department Report, June 1976. (To appear in *Algorithms and Complexity: New Directions and Recent Results*, ed. by J. F. Traub, Academic Press, New York, 1976.)

4. J.-C. Miellou, Itérations chaotiques à retards, *Comptes Rendus de l'Académie des Sciences de Paris*, Series A, Vol. 278, April 1974, pp. 957-960.

5. J. M. Ortega and W. C. Rheinboldt, *Iterative Solution of Nonlinear Equations in Several Variables*, Academic Press, New York, 1970.

6. F. Robert, M. Charnay, and F. Musy, Itérations chaotiques série-parallèle pour des équations non-linéaires de point fixe, *Aplicace Matematicky*, Vol. 20, 1975, pp. 1-38.

7. R. Varga, *Matrix Iterative Analysis*, Prentice Hall, Englewood Cliffs, New Jersey, 1962.

8. W. A. Wulf and C. G. Bell, C.mmp - A multi-mini-processor, *Proceedings of the AFIPS 1972 Fall Joint Computer Conference*, Vol. 41, December 1972, pp. 765-777.

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE		READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER	2. GOVT ACCESSION NO.	3. RECIPIENT'S CATALOG NUMBER
4. TITLE (and Subtitle) (6) ASYNCHRONOUS ITERATIVE METHODS FOR MULTIPROCESSORS		5. TYPE OF REPORT & PERIOD COVERED (9) Interim rept.
		6. PERFORMING ORG. REPORT NUMBER
7. AUTHOR(s) (10) Gerard M. Baudet		8. CONTRACT OR GRANT NUMBER(s) (15) N00014-76-C-0370, NSF-MCS-75-22255 NR 044-422
9. PERFORMING ORGANIZATION NAME AND ADDRESS Carnegie-Mellon University Computer Science Dept. Pittsburgh, PA 15213		10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS
11. CONTROLLING OFFICE NAME AND ADDRESS Office of Naval Research Arlingaton, VA 22217		12. REPORT DATE (11) November 1976
		13. NUMBER OF PAGES 33 (13) 33p.
14. MONITORING AGENCY NAME & ADDRESS (if different from Controlling Office)		15. SECURITY CLASS. (of this report) UNCLASSIFIED
		15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT (of this Report)

Approved for public release; distribution unlimited.

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

18. SUPPLEMENTARY NOTES

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

A class of asynchronous iterative methods is presented for solving a system of equations. Existing iterative methods are identified in terms of asynchronous iterations, and new schemes are introduced corresponding to a parallel implementation on a multiprocessor system with no synchronization between cooperating processes. A sufficient condition is given to guarantee the convergence of any asynchronous iterations, and results are extended to include iterative methods with memory.

over

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE S/N 0102-014-6601 |

UNCLASSIFIED 403 081

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

409

Asynchronous iterative methods are then evaluated from a computational point of view, and bounds are derived for the efficiency. The bounds are compared with actual measurements obtained by running various asynchronous iterations on a multiprocessor, and the experimental results show clearly the advantage of purely asynchronous iterative methods.